



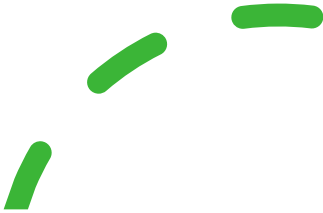
Object Oriented Programming - 5 Feature : Abstraction, Encapsulation, Inheritance, Polymorphism

PSTI – FT Unram





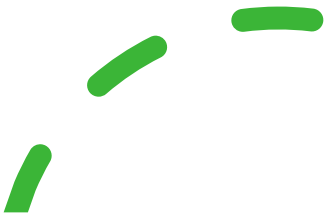
SOFTWARE : OBJECT ORIENTED

- Software that built from **classes** and **objects** that interact with each other.
 - Class
A static description of something.
 - Object
Something created (instantiation) of a class.
- 



Difference between Class and Object

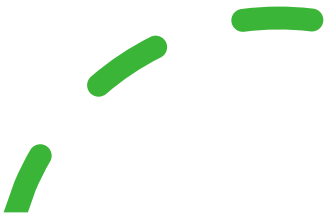
- Class : **concept** and **description** from something
 - Class declares **methods** that can be used (called) by objects
- Object : **an instance of a class** , a real form (example) of a class
 - Objects have **independent properties** and can be used to call methods
- Examples of Classes and Objects :
 - Class: **car**
 - Object: **Mr Joko's car, my car, red car**





Feature of Object Oriented Programming

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

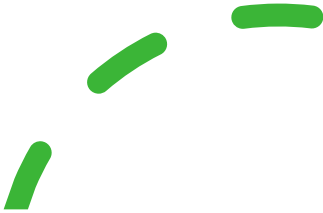




OOP Feature : Abstraction



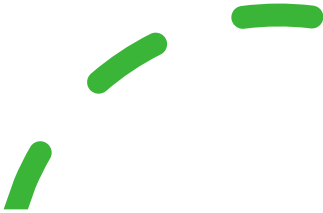
Abstraction (Design Level)

- Abstraction is a process of registering entities (objects) that will interact in an application.
 - These entities will later become classes, and from classes are born objects.
 - After the entities are registered, then the properties (state/attributes) and things that can be done by the entity (behaviour/method) are determined.
- 



CASE STUDY

Contoh masalah yang akan dibuat desain pemecahannya adalah sebagai berikut :

- sebuah koperasi menginginkan sebuah aplikasi pelaporan rekapitulasi stok barang yang mencatat persediaan, pemasukan dan pengeluaran barang. Laporan ini dilakukan seminggu sekali;
 - aplikasi ini akan dijalankan oleh seorang petugas khusus logistik. Petugas tersebut akan mencatat Stok barang didapat dari supplier-supplier, baik dari dalam maupun luar kota. Penjualan barang dilakukan pada sebuah mini market, yang pembelinya berasal dari anggota koperasi maupun yang bukan anggota koperasi;
 - kemudian ada kebijakan kalau barang berupa makanan yang masa kedaluwarsanya sudah lewat, harus dibuang, dan yang masa kedaluwarsanya tinggal 6 bulan lagi, diberikan diskon 50% (asumsi makananya semuanya mempunyai masa kedaluwarsa > 1 tahun). Kebijakan lainnya adalah : untuk item barang yang tinggal $\leq 40\%$ dari seharusnya, harus segera ditambah stoknya.
- 

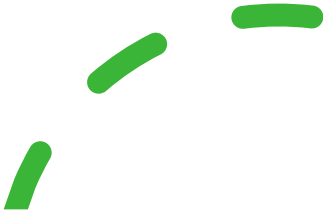



Entitas	Atribut	Operasi
Barang	<ul style="list-style-type: none"> – jumlah – harga beli – tanggal kadaluwarsa – harga jual – ID barang – diskon 	<ul style="list-style-type: none"> – hitung tanggal kadaluwarsa – hitung harga jual – hitung diskon
Supplier	<ul style="list-style-type: none"> – nama – lokasi – ID supplier – jenis barang 	<ul style="list-style-type: none"> – modifikasi nama supplier – modifikasi lokasi supplier
Koperasi	<ul style="list-style-type: none"> – daftar transaksi – gudang 	<ul style="list-style-type: none"> – modifikasi daftar transaksi
Pembeli	<ul style="list-style-type: none"> – ID pembeli – nama pembeli 	<ul style="list-style-type: none"> – .modifikasi ID pembeli – modifikasi nama pembeli
Gudang	<ul style="list-style-type: none"> – daftar stok barang – koperasi 	<ul style="list-style-type: none"> – modifikasi stok barang – set koperasi
Daftar Transaksi	<ul style="list-style-type: none"> – data penjualan [] – data pembelian [] 	<ul style="list-style-type: none"> – tambah data penjualan – tambah data pembelian
Daftar Stok Barang	<ul style="list-style-type: none"> – Barang [] – jumlahBarang[] 	<ul style="list-style-type: none"> – tambah barang – kurangi barang

SELEKSI ENTITAS



Seleksi entitas dilakukan dengan mempertimbangkan hal-hal berikut :

- Relevansi dengan permasalahan
 - Apakah entitas tersebut eksis pada batasan-batasan yang ditentukan pada permasalahan ?
 - Apakah entitas tersebut dibutuhkan untuk menyelesaikan masalah ?
 - Eksistensi entitas haruslah *independent*
- 

Hasil Seleksi Entitas

BARANG
jumlah harga beli tanggal kedaluwarsa harga jual ID barang
hitung tanggal kedaluwarsa hitung harga jual hitung diskon

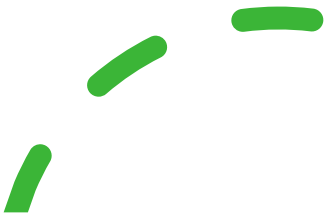
DAFTAR TRANSAKSI
data penjualan [] data pembelian []
tambah data penjualan tambah data pembelian

KOPERASI
daftar stok barang daftar transaksi
modifikasi daftar transaksi
DAFTAR STOK BARANG
barang [] jumlah barang []
tambah barang kurangi barang



Kesimpulan : Abstraksi (Design Level)

1. Menentukan domain sistem/aplikasi
2. Mendaftar entitas-entitas dalam sistem/aplikasi → Kelas
3. Mendata atribut dan behaviournya
4. Menseleksi entitas (kelas) yang pasti akan digunakan saja.

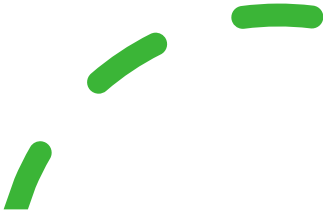




ABSTRACTION (Implementation Level)

1. Abstraction is about **hiding implementation details** and showing only the **essential features** of an object.
It answers: *"What does the object do?"*, not *"How does it do it?"*

2. In Java:

- Achieved using **abstract classes** and **interfaces**.
 - Clients interact with the abstract layer, without knowing implementation details.
- 



ABSTRACTION (Implementation Level)

```
// Abstract class
abstract class Vehicle {
    abstract void move(); // only defines behavior, not implementation
}

// Implementation
class Car extends Vehicle {
    @Override
    void move() {
        System.out.println("Car is driving");
    }
}

class Bicycle extends Vehicle {
    @Override
    void move() {
        System.out.println("Bicycle is pedaling");
    }
}
```

Hiding implementation.

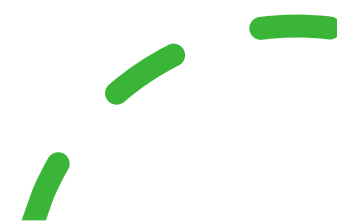
Users of Vehicle only need to know that it can move(). They don't care *how* each subclass implements it.



Enkapsulasi



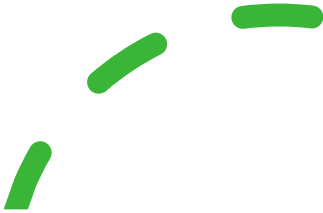
Enkapsulasi

- Enkapsulasi adalah suatu cara untuk menyembunyikan implementasi detail (berupa atribut atau method) dari suatu kelas.
 - Dalam OOP, kebutuhan akan enkapsulasi muncul karena adanya sharing data antar method
 - Dalam pemrograman, prinsip enkapsulasi dinyatakan dengan identifier private, default, protected, public.
 - Variabel/Atribut biasanya **private**, dan akses dilakukan via **getters and setters**.
- 



Encapsulation

```
public class BankAccount {  
    private double balance; // private field (hidden)  
  
    public BankAccount(double initialBalance) {  
        this.balance = initialBalance;  
    }  
  
    // public methods control access  
    public double getBalance() {  
        return balance;  
    }  
  
    public void deposit(double amount) {  
        if (amount > 0) {  
            balance += amount;  
        }  
    }  
}
```



The internal balance cannot be modified directly, only via controlled methods. This protects data integrity.

Struktur Dasar Kelas

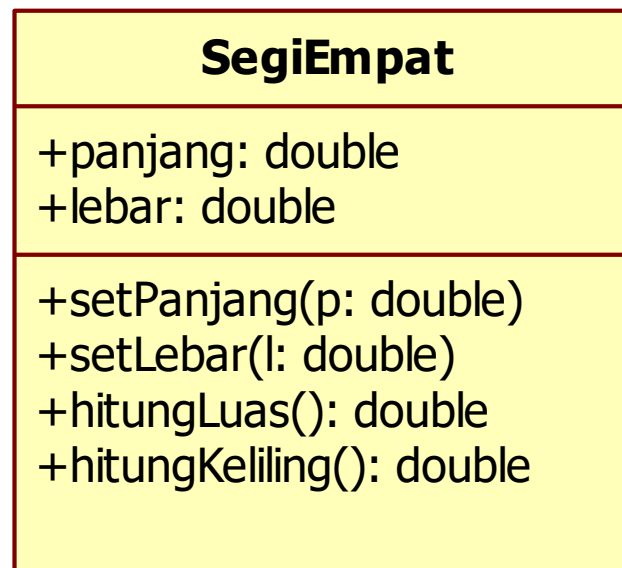
- Struktur dasar sebuah kelas pada intinya ada 4 bagian, yaitu :
 - deklarasi kelas;
 - deklarasi dan inisialisasi atribut;
 - pendefinisian *method*;
 - komentar.





PEMBUATAN KELAS

- Diagram UML (Unified Modeling Language)



Kode Kelas SegiEmpat

```
public class SegiEmpat{  
    //deklarasi variabel (atribut)  
    public double panjang;  
    public double lebar;  
  
    //deklarasi method  
    public void setPanjang(double p){  
        panjang = p;  
    }  
    public void setLebar(double l){  
        lebar = l;  
    }  
    public double hitungLuas(){  
        return panjang*lebar;  
    }  
    public double hitungKeliling(){  
        return 2*(panjang + lebar);  
    }  
}
```



KELAS UTAMA (MAIN)

```
public class SegiEmpat{
    public double panjang;
    public double lebar;
    public void setPanjang(double p){
        panjang = p;
    }
    public void setLebar(double l){
        lebar = l;
    }
    public double hitungLuas(){
        return panjang*lebar;
    }
    public double hitungKeliling(){
        return 2*(panjang + lebar);
    }
    public static void main(String[] args){
        SegiEmpat sel;
        sel = new SegiEmpat(); //pembentukan objek
        SegiEmpat se2 = new SegiEmpat(); // pembentukan objek
        sel.setPanjang(10);
        sel.setLebar(5);
        se2.setPanjang(5.5);
        se2.setLebar(2.3);
        System.out.println("Luas segi empat 1 =" +sel.hitungLuas());
        System.out.println("Keliling segi
        empat 2 =" +se2.hitungKeliling());
    }
}
```

SegiEmpat

+panjang: double
+lebar: double

+setPanjang(p: double)
+setLebar(l: double)
+hitungLuas(): double
+hitungKeliling(): double
+main(args: String[])



KELAS DENGAN KONSTRUKTOR

```
public class SegiEmpat{
    public double panjang;
    public double lebar;

    //deklarasi konstruktor
    public SegiEmpat(){
    }
    public SegiEmpat(double p, double l){
        panjang = p;
        lebar = l;
    }
    //deklarasi method
    public void setPanjang(double p){
        panjang = p;
    }
    public void setLebar(double l){
        lebar = l;
    }
    public double hitungLuas(){
        return panjang*lebar;
    }
    public double hitungKeliling(){
        return 2*(panjang + lebar);
    }
}
```

SegiEmpat
+panjang: double +lebar: double
+SegiEmpat() +SegiEmpat(p: double, l: double) +setPanjang(p: double) +setLebar(l: double) +hitungLuas(): double +hitungKeliling(): double +main(args: String[])

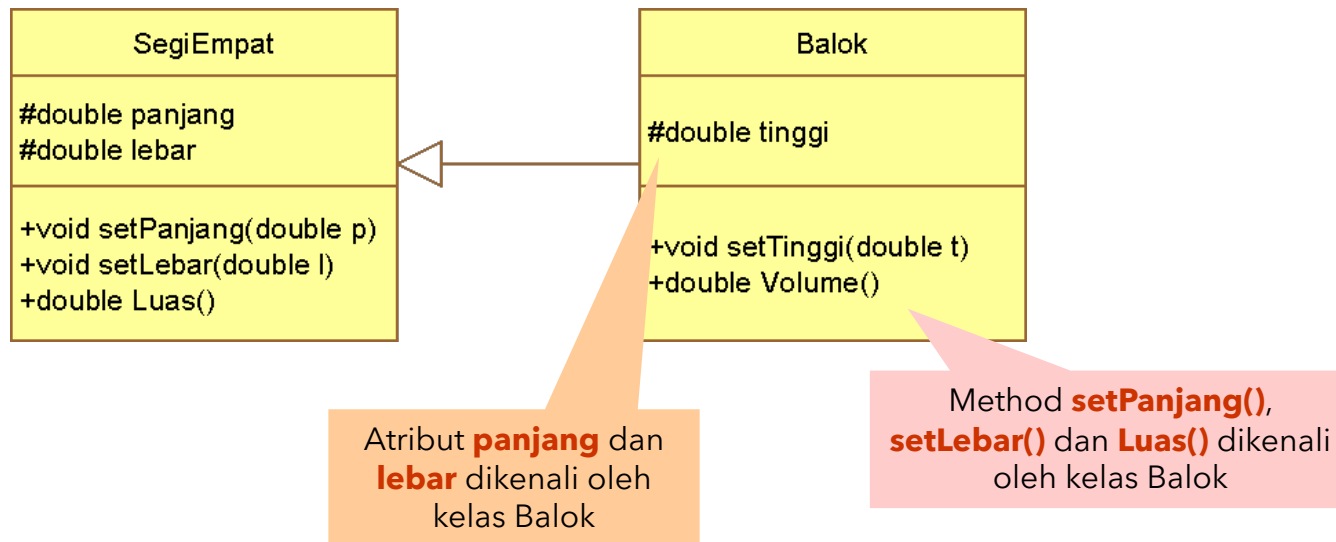
```
public static void main(String[] args){
    SegiEmpat sel;
    sel = new SegiEmpat(); //pembentukan
    objek
    sel.setPanjang(10);
    sel.setLebar(5);
    SegiEmpat se2 = new
    SegiEmpat(5.5,2.3); // pembentukan objek
    System.out.println("Luas segi empat 1
    =" +sel.hitungLuas());
    System.out.println("Keliling segi
    empat 2 =" +se2.hitungKeliling());
}
```

A stylized graphic on the left side of the slide. It features a solid olive-green circle representing a sun or moon, with several short, dashed olive-green lines above it suggesting rays or waves. The background is split into a light green upper half and a white lower half, separated by a curved line that follows the shape of the sun/moon.

Pewarisan

Apa yang Disebut Pewarisan ?

- Kemampuan sebuah kelas untuk **mewariskan** seluruh atau sebagian atribut dan methodnya ke kelas lain, sehingga atribut dan method tersebut **dikenal** oleh kelas yang menerima pewarisan **tanpa harus menuliskannya**.

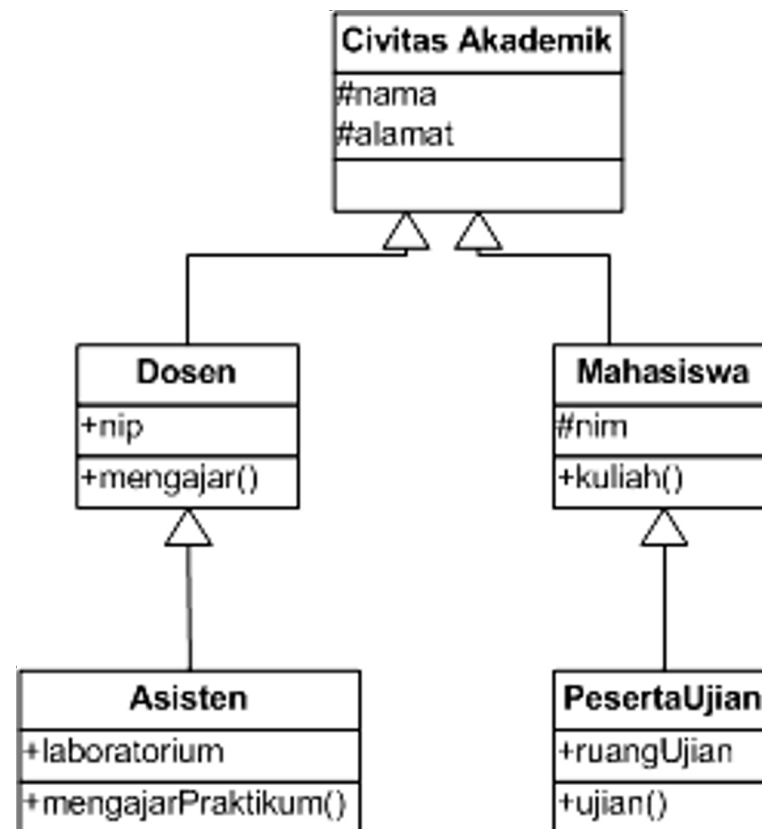




Apa yang Disebut Pewarisan ? (lanjutan)

- Kelas yang mewariskan disebut **kelas induk**, **super class**, atau **base class**.
- Kelas yang menerima pewarisan disebut **kelas anak**, **kelas turunan**, atau **subclass**.
- Merupakan implementasi dari relasi antar kelas **generalization-specialization** atau "**is-a**".

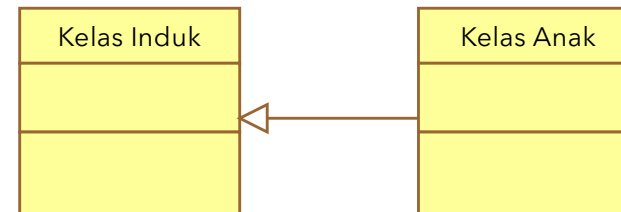
CONTOH PEWARISAN



Bentuk-bentuk Pewarisan

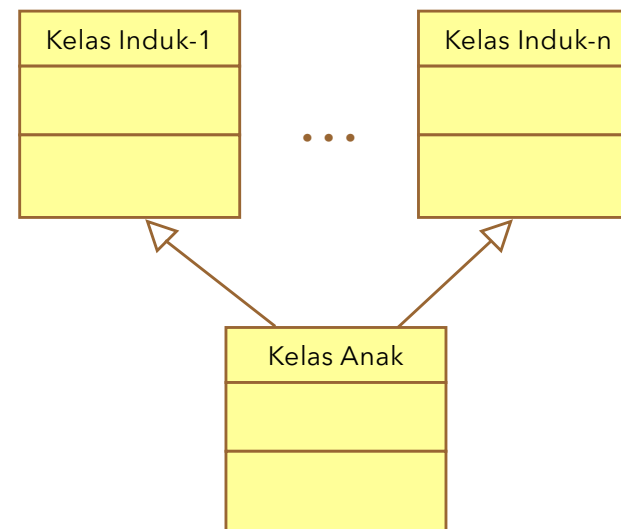
- Pewarisan Tunggal (*Single Inheritance*)

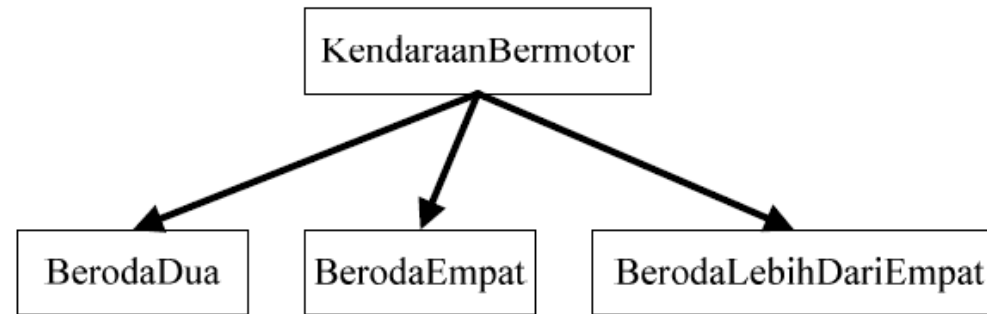
Kelas anak menerima pewarisan dari **satu** kelas induk.



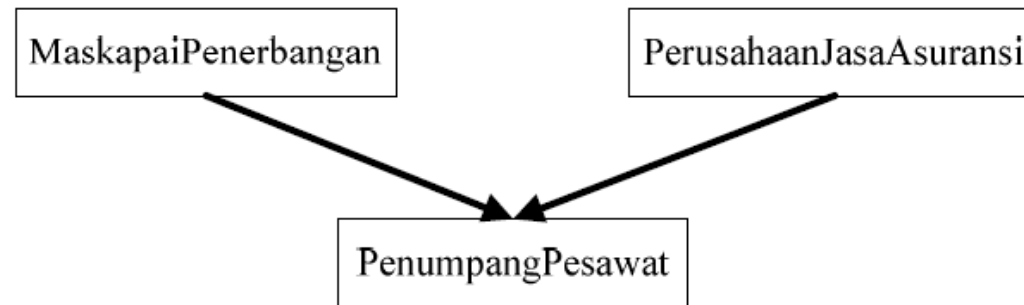
- Pewarisan Majemuk (*Multiple Inheritance*)

Kelas anak menerima pewarisan dari **beberapa** kelas induk. Bahasa Java tidak mengakomodasi *multiple inheritance*





Gambar 1-7 *Single Inheritance*



Gambar 1-8 *Multiple Inheritance*



Pewarisan dalam Java

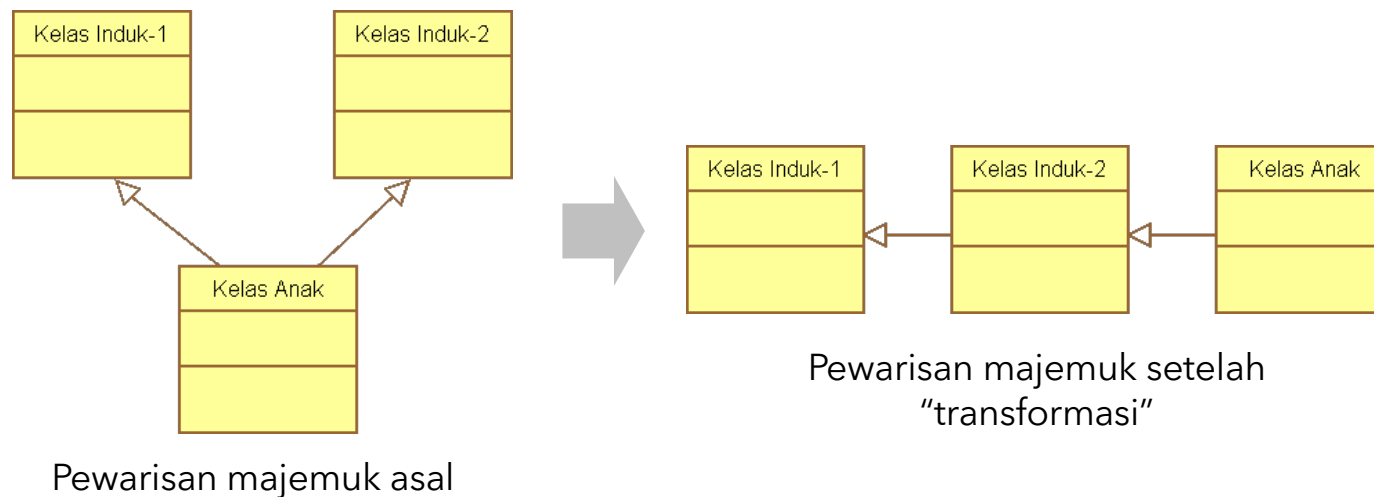
- Dinyatakan dengan menggunakan kata kunci (keyword) **extends**:

```
class KelasAnak extends KelasInduk {  
    // Atribut kelas anak  
    // Method kelas anak  
}
```

- Semua atribut dan method dengan *access modifier* **non-private** akan **diwariskan** dari kelas induk ke kelas anak.
- Method yang diwariskan dapat didefinisikan ulang di kelas anak (*override*) → polymorphism.

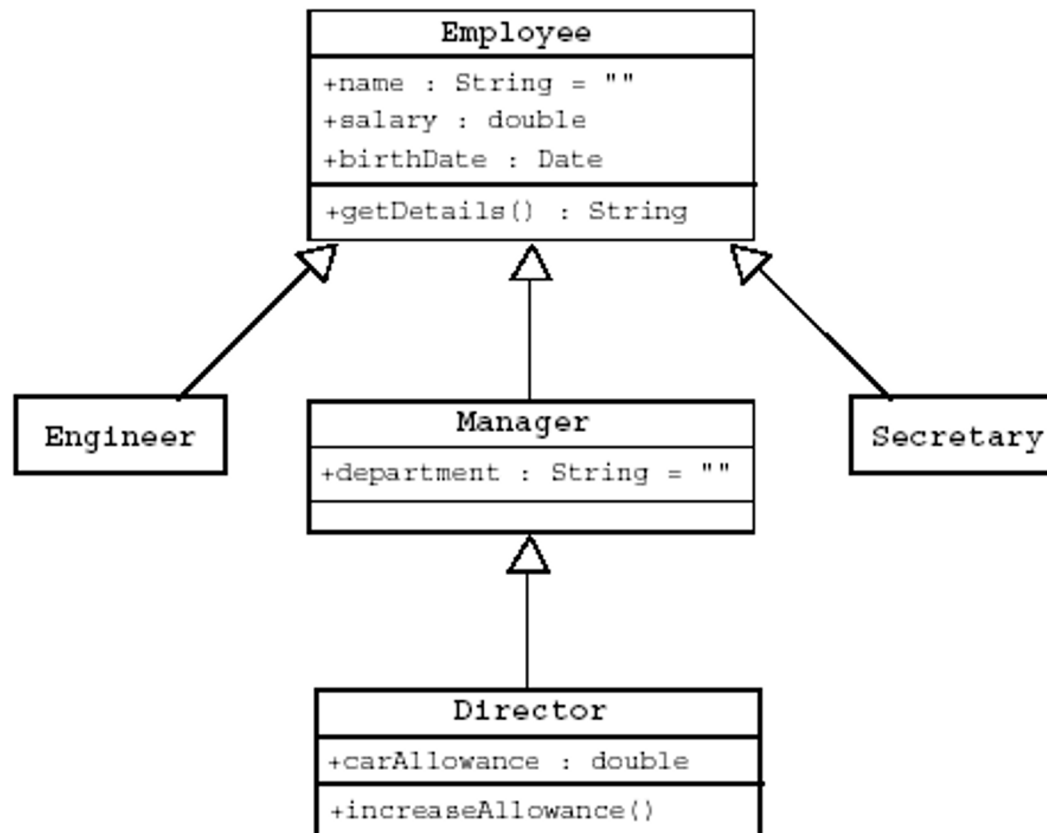
Pewarisan dalam Java (lanjutan)

- Hanya memperbolehkan pewarisan tunggal.
- Pewarisan majemuk harus dinyatakan sebagai pewarisan tunggal yang ditulis secara berjenjang.



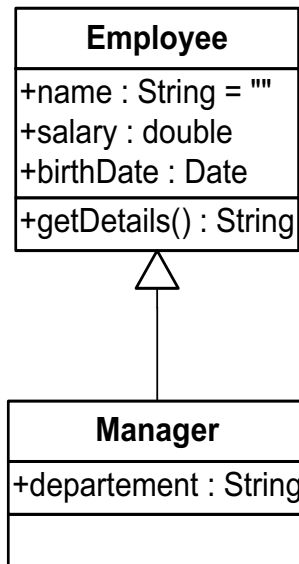
Pewarisan dalam Java (lanjutan)

- Single dan multilevel inheritance



Contoh Program Pewarisan

- Contoh 1



```
public class Employee {
    public String name;
    public Date birthDate;
    public double salary;

    public String getDetails() {...}
}

public class Manager extends
Employee {
    public String department;
}
```

Sepeda.java

```
public class Sepeda{  
    public int gir;  
  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir() {  
        return gir;  
    }  
}
```


Class SepedaGunung Mewarisi Class Sepeda

```
public class SepedaGunung extends
    Sepeda{

    private int sadel;

    void setSadel (int jumlah) {
        sadel = getGir() - jumlah;
    }

    int getSadel(){
        return sadel;
    }
}
```

SepedaGunung.java

```
public class SepedaGunungBeraksi {
    public static void main(String[] args) {

        SepedaGunung sg=new SepedaGunung();

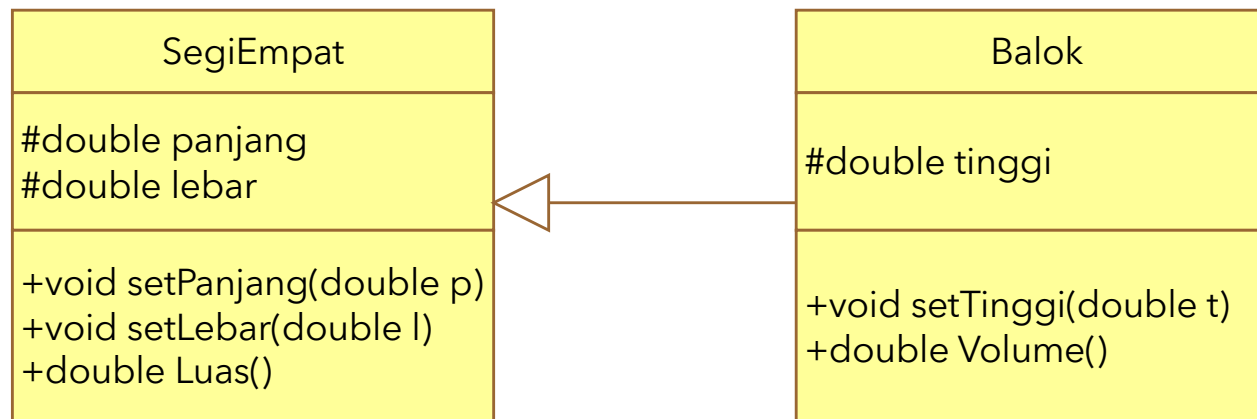
        sg.setGir(3);
        System.out.println(sg.getGir());

        sg.setSadel(1);
        System.out.println(sg.getSadel());
    }
}
```

SepedaGunungBeraksi.java

Contoh Program Pewarisan

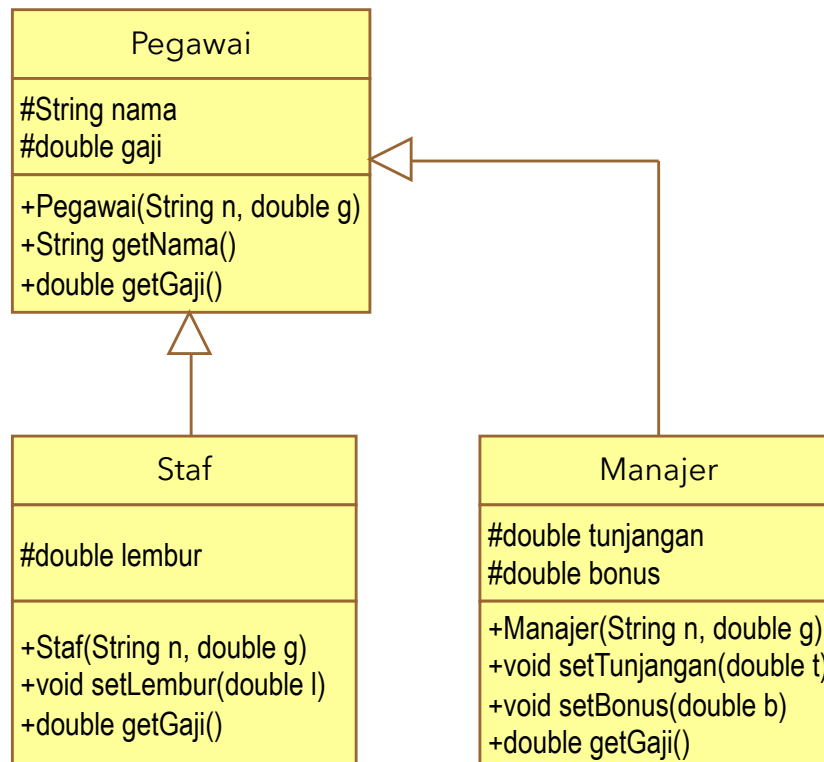
- Contoh 2



- [SegiEmpat.java](#)
- [Balok.java](#)
- [Driver.java](#)

Contoh Program Pewarisan (lanjutan)

- Contoh 3

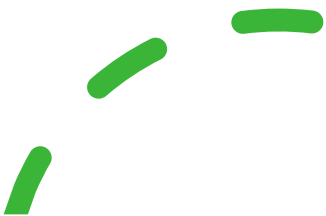


- [Pegawai.java](#)
- [Staf.java](#)
- [Manajer.java](#)
- [Driver.java](#)



Kapan Pewarisan ?

- Ada beberapa atribut dan method yang sama yang digunakan oleh beberapa kelas berbeda (reduksi penulisan kode).
- Ada satu atau beberapa kelas yang sudah pernah dibuat yang dibutuhkan oleh aplikasi (*reusability*).
- Ada perubahan kebutuhan fungsional atau *feature* aplikasi dimana sebagian atau seluruh perubahan tersebut tercakup di satu atau beberapa kelas yang sudah ada (*extend*).





Keyword `super`

- `super` digunakan untuk merefer superclass dari suatu class, yaitu untuk merefer member dari superclass, baik data attributes maupun methods



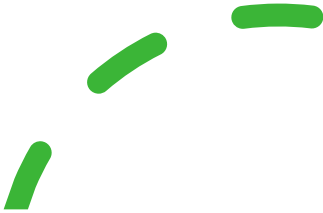


Pewarisan dalam Java (lanjutan)

- Untuk memanggil konstruktor kelas induk dari (konstruktor) kelas anak digunakan notasi **super()**, dan harus ditulis di baris pertama pada konstruktor kelas anak.

```
public Balok() {  
    super();  
    tinggi = 0;  
}
```

- Untuk memanggil method kelas induk dari kelas anak digunakan notasi **super.namaMethod()**.



```
public double getGaji() {  
    double pokok = super.getGaji();  
    return (pokok+lembur);  
}
```

Keyword super



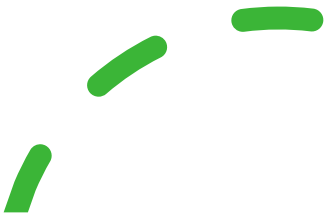
```
1.class Employee {  
2.  public String name;  
  
3.  public Employee (String s) {  
4.      name = s  
5.  }  
6.}
```

```
1.class Manager extends Employee {  
2.  private String alamat;  
3.  public Manager(String nama, String s) {  
4.      super.name = nama ;  
5.      alamat = s;  
6.  }  
7.}
```



Konstruktor tidak diwariskan

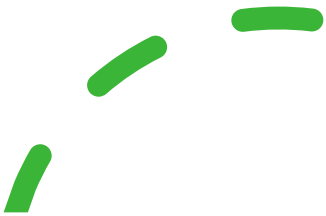
- Konstruktor dari parent class tidak dapat diwariskan ke subclass-nya.
- Konsekuensinya, setiap kali kita membuat suatu subclass, maka kita harus memanggil konstruktor parent class.
- Pemanggilan konstruktor parent harus dilakukan pada baris pertama dari konstruktor subclass.





Konstruktor tidak diwariskan

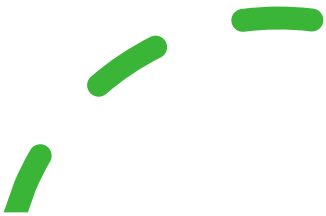
- Jika kita tidak mendeklarasikannya secara eksplisit, maka kompiler Java akan menambahkan deklarasi pemanggilan konstruktor parent class di konstruktor subclass.





Konstruktor tidak diwariskan

- Sebelum subclass menjalankan konstruktornya sendiri, subclass akan menjalankan constructor superclass terlebih dahulu.
- Hal ini terjadi karena secara implisit pada constructor subclass ditambahkan pemanggilan `super()` yang bertujuan memanggil constructor superclass oleh kompiler.

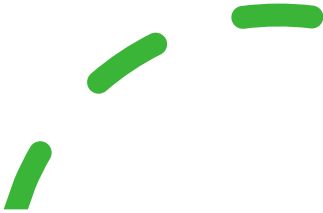




Misalnya saja kita mempunyai dua buah class sebagai berikut :

```
public class Parent  
{  
  
}
```

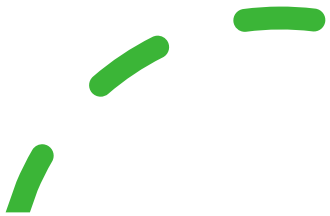
```
public class Child extends Parent {  
  
}
```





Contoh

- Pada saat program tersebut dikompilasi, maka kompiler Java akan menambahkan :
 - konstruktor class Parent
 - konstruktor class Child
 - pemanggilan konstruktor class Parent di konstruktor class Child






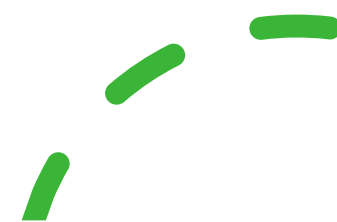
Sehingga program tersebut sama saja dengan yang berikut ini :

```
public class Parent {  
    public Parent() {  
  
    }  
}
```

```
public class Child extends Parent {  
    public Child() {  
        super();  
    }  
}
```



pemanggilan kostruktor class Parent



Contoh

```
public class Child extends Parent {  
    int x;  
    public Child() {  
        x = 5;  
        super();  
    }  
}
```

X

```
public class Child extends Parent {  
    int x;  
    public Child() {  
        super();  
        x = 5;  
    }  
}
```

✓

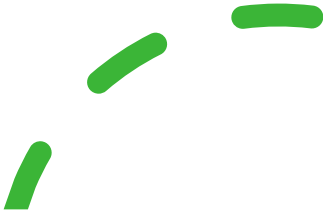


Constructor pada inheritance

- Misalkan kita buat class parent bernama Person sbb :

```
public class Person {  
    protected String name;  
    protected String address;  
  
    public Person(){  
        System.out.println("Inside Person:Constructor");  
    }  
}
```

- Sekarang, kita buat class lain bernama Student yang meng-extends class Person.

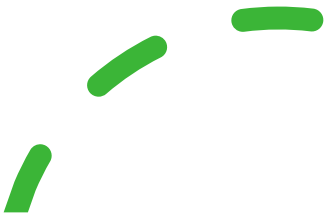


```
public class Student extends Person {  
    public Student(){  
        System.out.println("Inside Student:Constructor");  
    }  
}
```



Alur Eksekusi Constructor

- Ketika sebuah object Student diinstansiasi, *default constructor* dari superclass Student dipanggil secara implisit untuk melakukan inisialisasi seperlunya.
- Setelah itu, pernyataan di dalam constructor subclass baru dijalankan.





Penjelasan

- Untuk memperjelasnya, perhatikan kode dibawah ini,

```
public static void main( String[] args ){  
    Student anna = new Student();  
}
```

- Dalam kode ini, kita membuat sebuah object dari class Student. Hasil dari program adalah:

```
Inside Person:Constructor  
Inside Student:Constructor
```

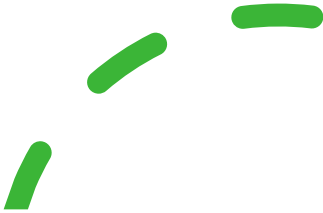
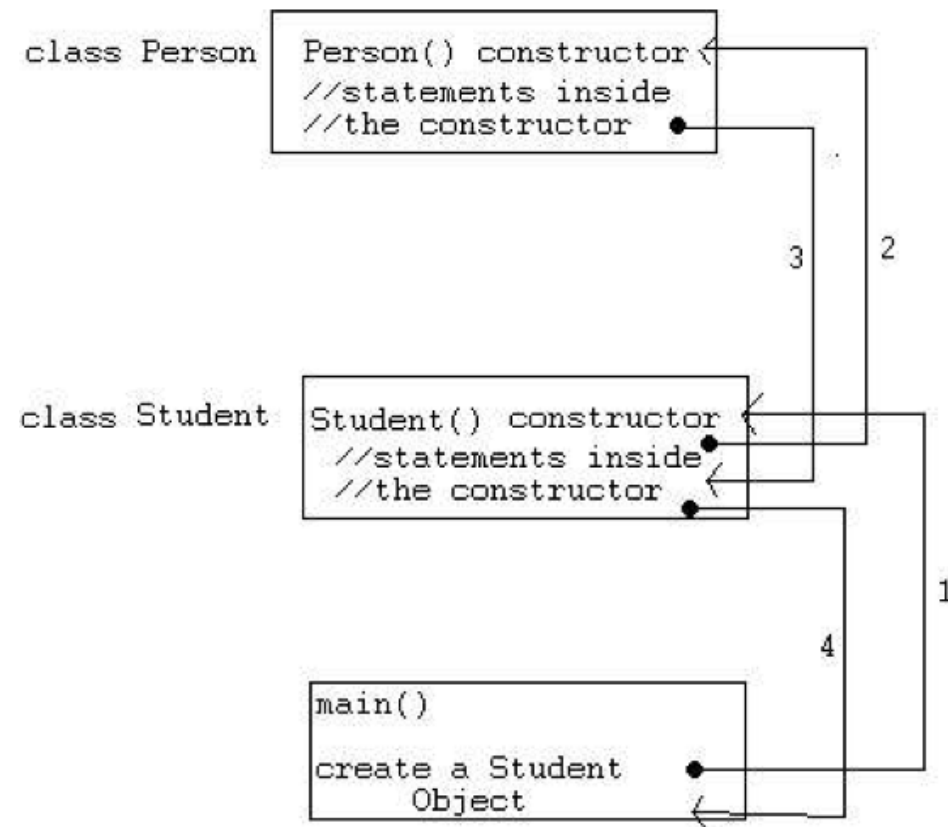



Diagram Alur Eksekusi Constructor





Superclass Constructor	Subclass Code	Compiler Action	Result
✓ Has no-arg constructor (default or explicit)	Subclass defines constructor without <code>super()</code>	Compiler inserts <code>super();</code>	✓ Compiles, superclass constructor runs first
✓ Has parameterized constructor(s) only (no no-arg)	Subclass defines constructor without <code>super(args)</code>	Compiler inserts <code>super();</code> but none exists	✗ Compile-time error
✓ Has parameterized constructor(s)	Subclass defines constructor and explicitly calls <code>super(args)</code>	Explicit <code>super(args)</code> matches superclass constructor	✓ Compiles, chosen parent constructor runs first
✗ Superclass has no constructor at all	(Java provides default no-arg constructor automatically)	Compiler inserts <code>super();</code>	✓ Compiles (default no-arg is created automatically)

Constructor Behavior in Java



If superclass has a **no-arg constructor**, you're safe (implicit `super()` works).



If superclass has **only parameterized constructors**, subclass must explicitly call one of them.

If superclass has **no constructor defined**, Java provides a hidden no-arg one.

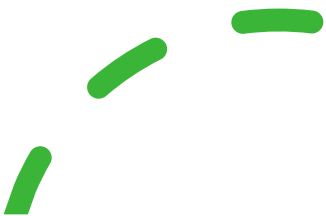




1. Superclass with no-arg constructor

```
class Parent {  
    Parent() { System.out.println("Parent no-arg"); }  
}  
class Child extends Parent {  
    Child() { System.out.println("Child"); }  
}  
new Child();
```

Output:
Parent no-arg
Child



2. Superclass with only parameterized constructor (Error)

```
class Parent {  
    Parent(String msg) { System.out.println(msg); }  
}  
class Child extends Parent {  
    Child() { System.out.println("Child"); } // ✖ Error  
}
```

Output:

Compile error: constructor Parent in class Parent cannot be applied to given types.



3. Parameterized constructor with explicit `super(args)`

```
class Parent {  
    Parent(String msg) { System.out.println("Parent: " + msg); }  
}  
class Child extends Parent {  
    Child() { super("Hello"); System.out.println("Child"); }  
}  
new Child();
```

Output:
Parent: Hello
Child





4. No constructor defined in superclass

```
class Parent { } // compiler generates default no-arg constructor
class Child extends Parent {
    Child() { System.out.println("Child"); }
}
new Child();
```

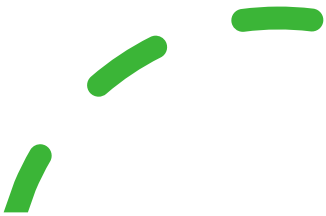
Output:
Child





Kontrol pengaksesan

- Dalam dunia riil, suatu entitas induk bisa saja tidak mewariskan sebagian dari apa-apa yang ia punyai kepada entitas turunan karena sesuatu hal.
- Demikian juga dengan konsep inheritance dalam OOP.
- Suatu parent class dapat tidak mewariskan sebagian member-nya kepada subclass-nya.
- Sebagai contoh, kita coba untuk memodifikasi class Pegawai.
- Hal ini dipengaruhi oleh **access modifier**.



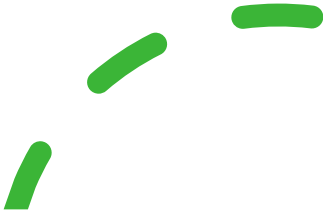
Hak Akses

Modifier	Kelas yang sama	Kelas Turunan	Object dlm Package yg sama	Object yg berbeda package
Private	V	X	X	X
Protected	V	V	V	X
Public	V	V	V	V



Contoh

```
public class Pegawai {  
    private String nama;  
    public double gaji;  
}
```





Contoh

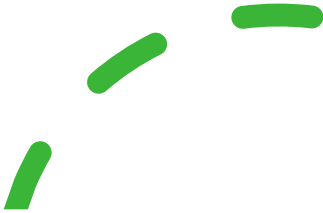
- Coba untuk mengkompilasi class Manajer pada contoh sebelumnya.
- Apa yang terjadi?
- Pesan kesalahan akan muncul seperti ini :

```
Manajer.java:5: nama has private access in Pegawai
    nama=n;
```

- Ini membuktikan bahwa class Manajer tidak mewarisi data member nama dari parent class-nya (Pegawai).
- 



protected

- Protected mempunyai kemampuan akses yang lebih besar daripada private dan default.
 - Protected feature dari suatu class bisa diakses oleh semua class dalam satu package.
- 

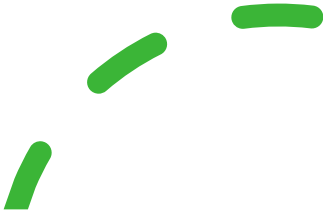
Example: protected

```
1. package adifferentpackage; // Class Ski now in
// a different package
2. Public class Ski {
3.     protected void applyWax() { . . . }
4. }
```

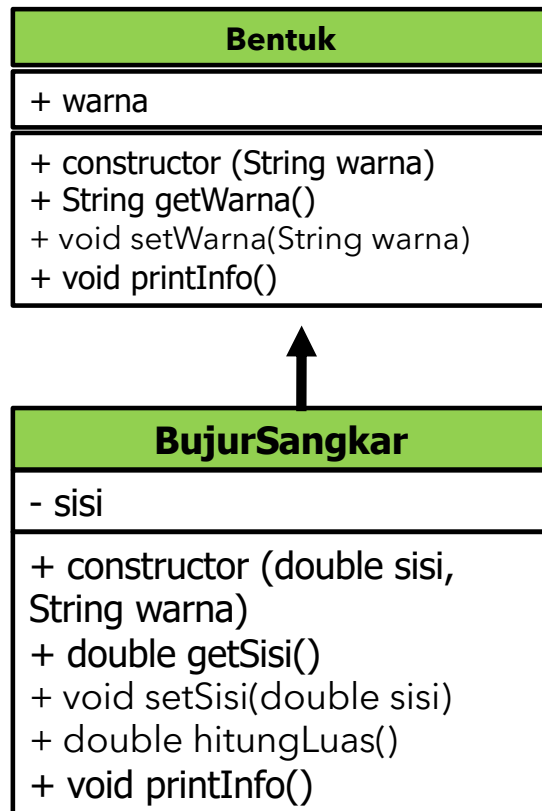
```
1. Public package sportinggoods;
2. class DownhillSki extends Ski {
3.     void tuneup() {
4.         applyWax(); → OK
5.         // other tuneup functionality here
6.     }
7. }
```



Latihan: Inheritance Matematika

1. Buat class **MatematikaCanggih** yang merupakan **inherit** dari class **Matematika**
 1. Tambahkan method **modulus(int a, int b)** yang menghitung modulus dari a dan b
 2. Operator modulus adalah **%**
 2. Buat class **MatematikaCanggihBeraksi** yang memanggil method pertambahan, perkalian dan modulus
- 

Latihan 1



Untuk kelas Bentuk :

- **getWarna** adl method yang akan mengembalikan nilai variabel warna
- **setWarna** adl method untuk mengubah nilai variabel warna
- **printInfo** adl method yang akan menuliskan "Bentuk berwarna [warna]"

Untuk kelas BujurSangkar :

- **getSisi** adl method yang akan mengembalikan nilai variabel sisi
- **setSisi** adl method untuk mengubah nilai variabel sisi
- **hitungLuas** adl method yang akan mengembalikan hasil perhitungan luas bujursangkar
- **printInfo** adl method yang akan menuliskan "Bujursangkar berwarna [warna], luas = [luas]"

Latihan 2

- Buatlah kelas Lingkaran sbg turunan kelas Bentuk.

Lingkaran
- radius
+ constructor (double radius, String warna) + double getRadius() + void setRadius(double r) + double hitungLuas() + void printInfo()

- **constructor** akan menginisialisasi radius dan warna
- **getRadius** adl method yang akan mengembalikan nilai variabel radius
- **setRadius** adl method untuk mengubah nilai variabel radius
- **hitungLuas** adl method yang akan mengembalikan hasil perhitungan luas lingkaran (PHI jadikan sbg konstanta kelas)
- **printInfo** adl method yang akan menuliskan "Lingkaran [warna], luas = [luas]"

Latihan 3

- Buatlah kelas Silinder sbg turunan kelas Lingkaran

Silinder
- tinggi
+ constructor (double tinggi, double radius, String warna) + double getTinggi() + void setTinggi(double t) + double hitungVolume() + void printInfo()

- **constructor** akan menginisialisasi variabel tinggi, radius, dan warna
- **getTinggi** adl method yg akan mengembalikan tinggi
- **setTinggi** adl method yg akan mengubah tinggi
- **hitungVolume** adl method yg akan mengembalikan hasil perhitungan volum silinder
- **printInfo** adl method yg akan menuliskan "Silinder warna [warna], volume = [volume]"



Polymorphism

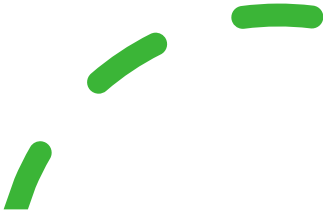
Polymorphism

- *Polymorphism* memungkinkan suatu method memiliki bentuk yang berbeda
- *Polymorphism* ada 2 jenis :
 - Overloading
 - Override





Polymorphism

- Kemampuan untuk **memperlakukan object yang memiliki perilaku (bentuk) yang berbeda**
 - Implementasi konsep polymorphism:
 1. **Overloading**: Kemampuan untuk menggunakan **nama yang sama** untuk beberapa **method yang berbeda parameter (tipe dan atau jumlah)**
 2. **Overriding**: Kemampuan subclass untuk **menimpa method** dari superclass, yaitu dengan cara menggunakan nama dan parameter yang sama pada method
- 

Polymorphism – Overloading

```
class Mobil {
    String warna;
    int tahunProduksi;

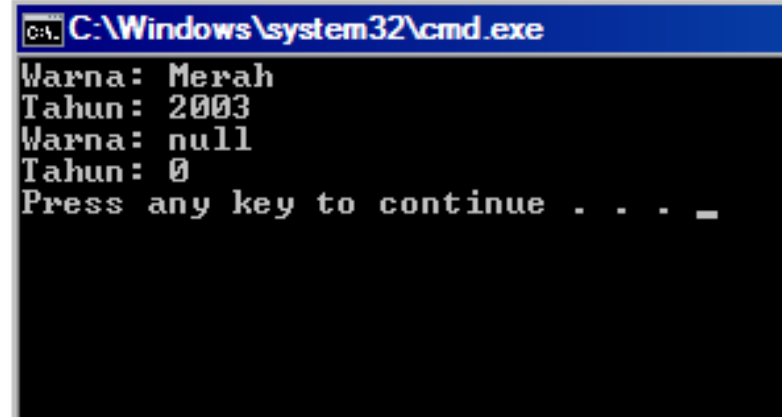
    public Mobil(String warna, int
tahunProduksi){
        this.warna = warna;
        this.tahunProduksi = tahunProduksi;
    }

    public Mobil(){
    }

    void info(){
        System.out.println("Warna: " + warna);
        System.out.println("Tahun: " + tahunProduksi);
    }
}
```

```
public class MobilKonstruktor{
    public static void main(String[] args){
        Mobil mobilku = new Mobil("Merah", 2003);
        mobilku.info();

        Mobil mobilmu = new Mobil();
        mobilmu.info();
    }
}
```



```
C:\Windows\system32\cmd.exe
Warna: Merah
Tahun: 2003
Warna: null
Tahun: 0
Press any key to continue . . . _
```

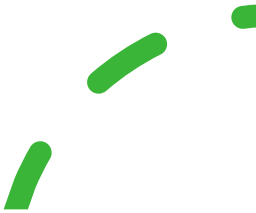
Polymorphism – Overloading

```
class Lingkaran{
    void gambarLingkaran(){
    }
    void gambarLingkaran(int diameter){
    ...
    }
    void gambarLingkaran(double diameter){
    ...
    }
    void gambarLingkaran(int diameter, int x, int y){
    ...
    }
    void gambarLingkaran(int diameter, int x, int y, int warna, String
namaLingkaran){
    ...
    }
}
```

Polymorphism - Overriding



```
public class Sepeda{  
    private int gir;  
  
    void setGir(int pertambahanGir){  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir(){  
        return gir;  
    }  
}
```



Polymorphism - Overriding

```
public class SepedaGunung extends
    Sepeda{

    void setGir(int pertambahanGir) {
        super.setGir(pertambahanGir);
        gir = 2*getGir();
    }
}
```

SepedaGunung.java

```
public class SepedaGunungBeraksi {
    public static void main(String[] args) {

        SepedaGunung sg=new
            SepedaGunung();

        sg.setGir(2);
        System.out.println(sg.getGir());

        sg.setGir(3);
        System.out.println(sg.getGir());
    }
}
```

SepedaGunungBeraksi.java

Latihan: Overloading pada Matematika

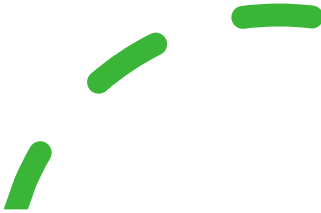
1. Kembangkan class **Matematika**, **MatematikaCanggih** dan **MatematikaBeraksi**
2. Lakukan **overloading** pada **Method** yang ada (pertambahan, pengurangan, perkalian, pembagian, modulus)
3. Tambahkan method baru bertipe data **double** (pecahan) dan **memiliki 3 parameter**
4. Uji di kelas **MatematikaBeraksi** dengan parameter pecahan: 12.5, 28.7, 14.2
5. Uji konsep overloading dengan:
`pertambahan(12.5, 28.7, 14.2)`
`pertambahan(12, 28, 14)`
`pertambahan(23, 34)`
`pertambahan(3.4, 4.9)`



Overloading Constructor

```
1. public class Employee {
2.     private static final double BASE_SALARY = 15000.00;
3.     private String name;
4.     private double salary;
5.     private Date birthdate;

6.     public Employee (String name, double salary, Date DoB) {
7.         this.name = name;
8.         this.salary = salary;
9.         birthdate = DoB;
10.    }
11.    public Employee(String name, double salary) {
12.        this(name,salary,null);
13.    }
14.    public Employee(String name, Date DoB) {
15.        this(name,BASE_SALARY,DoB);
16.    }
17.    public Employee(String name) {
18.        this(name,BASE_SALARY);
19.    }
20.    //more Employee code...
21. }
```



Overloading Method

```
1. public class Employee {
2.     private static final double BASE_SALARY = 15000.00;
3.     private String name;
4.     private double salary;
5.     private Date birthdate;

6.     public Employee (String name, double salary, Date DoB) {
7.         this.name = name;
8.         this.salary = salary;
9.         birthdate = DoB;
10.    }
11.    public void set_Employee(String name, double salary) {
12.        this(name,salary,null);
13.    }
14.    public void set_Employee(String name, Date DoB)      {
15.        this(name,BASE_SALARY,DoB);
16.    }
17.    public void set_Employee(String name)      {
18.        this(name,BASE_SALARY);
19.    }
20.    //more Employee code...
21. }
```

Overriding Methods



- Sub class bisa mendefinisikan ulang suatu method yang sudah diwariskan dari parent class
- Sub class bisa mendefinisikan suatu method yang secara fungsionalitas berbeda dengan method yang diwariskan dari parent class, dengan syarat memiliki kesamaan :
 - Nama Method
 - Return Type
 - Argument list

Overriding Methods



```
1. class Employee {
2.     private String name;
3.     private MyDate birthDate;
4.     private float salary;

5.     public String getDetails() {
6.         return "Name: " + name + "\nSalary: " + salary
7.             + "\nBirth Date: " + birthDate;
8.     }
9. }

1. class Manager extends Employee {
2.     protected String department;

3.     public String getDetail() {
4.         return super.getDetails() + "\nDepartment: " +
5.             department;
6.     }
7. }
```

Penerapan Pewarisan

```
12 public class Kendaraan {
13     private String nama_pemilik;
14     protected String nomor_mesin;
15     public int harga;
16
17     public Kendaraan() {
18         nama_pemilik = "Negara";
19         nomor_mesin = "0";
20         harga = 0;
21         System.out.println("Ctor Kendaraan");
22     }
23
24     public Kendaraan(String pemilik, String mesin, int price) {
25         nama_pemilik = pemilik;
26         nomor_mesin = mesin;
27         harga = price;
28         System.out.println("Copy Ctor Kendaraan");
29     }
30
31     public void deskripsi() {
32         System.out.println("Kendaraan ini milik " + nama_pemilik);
33         System.out.println("Nomor Mesin kendaraan ini " + nomor_mesin);
34         System.out.println("Harga kendaraan ini = " + harga);
35     }
36
37     public void identitas() {
38         System.out.println("Ini kendaraan");
39     }
40 }
```

```
11 public class Motor extends Kendaraan{
12     public Motor() {
13         System.out.println("Ctor Motor");
14         System.out.println("Harga Motor = " + harga); // bisa
15         System.out.println("Nomor Mesin Motor = " + nomor_mesin); // bisa
16         //System.out.println("Pemilik Motor = " + nama_pemilik); // TIDAK bisa
17     }
18 }
```

```
17 public static void main(String[] args) {
18     // TODO code application logic here
19     Motor m1 = new Motor();
20     m1.deskripsi();
21 }
```

```
Output - Konsep_PBO (run)
run:
Ctor Kendaraan
Ctor Motor
Harga Motor = 0
Nomor Mesin Motor = 0
Kendaraan ini milik Negara
Nomor Mesin kendaraan ini 0
Harga kendaraan ini = 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penerapan Pewarisan

```
11 public class Motor extends Kendaraan{
12     public Motor() {
13         System.out.println("Ctor Motor");
14         System.out.println("Harga Motor = " + harga); // bisa
15         System.out.println("Nomor Mesin Motor = " + nomor_mesin); // bisa
16         //System.out.println("Pemilik Motor = " + nama_pemilik); // TIDAK bisa
17     }
18 }
```

```
17 public static void main(String[] args) {
18     // TODO code application logic here
19     Motor m1 = new Motor();
20     m1.deskripsi();
21 }
```

Output - Konsep_PBO (run)

```
run:
Ctor Kendaraan
Ctor Motor
Harga Motor = 0
Nomor Mesin Motor = 0
Kendaraan ini milik Negara
Nomor Mesin kendaraan ini 0
Harga kendaraan ini = 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penerapan Polymorphism

```
12 public class Kendaraan {  
13     private String nama_pemilik;  
14     protected String nomor_mesin;  
15     public int harga;  
16     public Kendaraan() {...}  
22  
23     public Kendaraan(String pemilik, String mesin, int price) {...}  
29  
30     public void deskripsi() {...}  
35  
36     public void identitas() {  
37         System.out.println("Ini kendaraan");  
38     }  
39 }
```

```
11 public class Motor extends Kendaraan {  
12     public Motor() {...}  
18  
19     @Override  
20     public void identitas() {  
21         System.out.println("Ini Motor");  
22     }  
23 }
```

```
11 public class Mobil extends Kendaraan {  
12  
13     @Override  
14     public void identitas() {  
15         System.out.println("Ini Mobil");  
16     }  
17 }
```


Penerapan Polymorphism

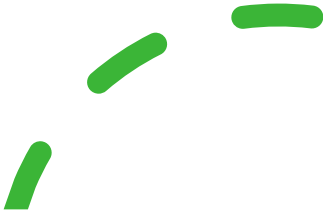
```
17 public static void main(String[] args) {  
18     // TODO code application logic here  
19     Kendaraan k = new Kendaraan();  
20     Kendaraan mtr = new Motor();  
21     Kendaraan mbl = new Mobil();  
22  
23     k.identitas();  
24     mtr.identitas();  
25     mbl.identitas();  
26 }
```

Output - Konsep_PBO (run)

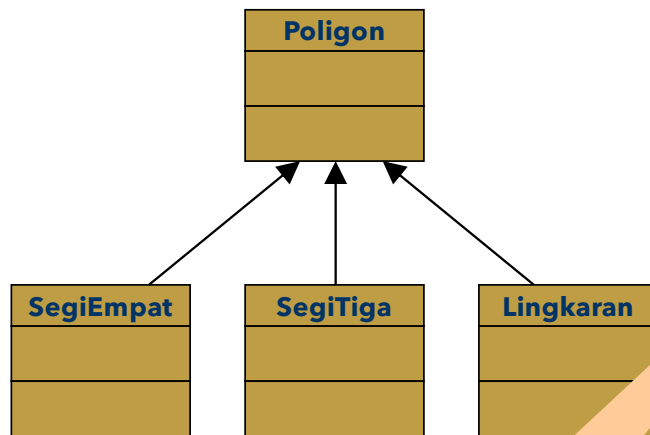
```
run:  
Ctor Kendaraan  
Ctor Kendaraan  
Ctor Motor  
Harga Motor = 0  
Nomor Mesin Motor = 0  
Ctor Kendaraan  
Ini kendaraan  
Ini Motor  
Ini Mobil  
BUILD SUCCESSFUL (total time: 0 seconds)
```



Apa yang Disebut Polimorfisme ? (lanjutan)

- Kemampuan suatu objek untuk digunakan di banyak tujuan berbeda dengan nama yang sama.
 - Kemampuan objek dalam memberikan respon yang berbeda terhadap message yang mempunyai nama yang sama.
 - Disebut juga:
 - *dynamic binding*
 - *late binding*
 - *run-time binding*
- 

Gambaran Polimorfisme



variabel **p** menunjuk
objek yang diciptakan
dari kelas **Poligon**

variabel **p** diubah sehingga
menunjuk **objek** dari kelas
SegiEmpat

```
// Program Utama
class TesPoligon {
    Poligon p = new Poligon();
    ...

    SegiEmpat s = new SegiEmpat();
    SegiTiga t = new SegiTiga();
    Lingkaran l = new Lingkaran();

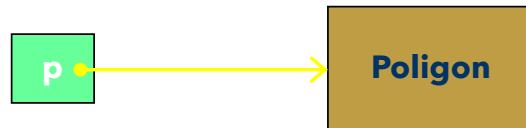
    p = s; // ok krn instan kelas
           anak juga instan kelas induk
    ...
}
```

variabel **s** menunjuk
objek yang diciptakan
dari kelas **SegiEmpat**

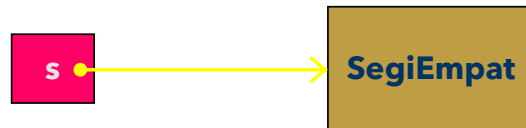
Gambaran Polimorfisme (lanjutan)

Sebelum:

```
Poligon p = new Poligon();
```

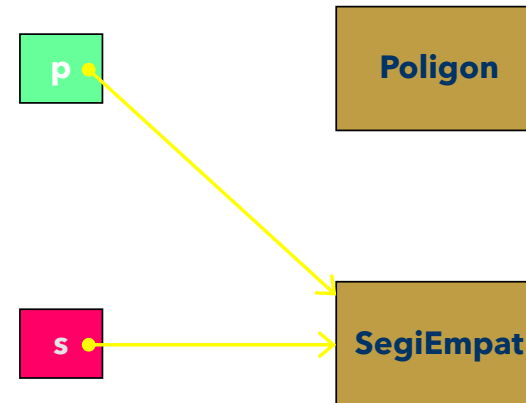


```
SegiEmpat s = new SegiEmpat();
```



Sesudah:

```
p = s;
```



Gambaran Polimorfisme (lanjutan)

```
class SegiEmpat {  
    . . .  
    public String display() {  
        return ("Segi Empat"  
            + "\no Panjang : " + getPanjang()  
            + "\no Lebar : " + getLebar()  
            + "\no Luas : " + Luas()  
            + "\no Keliling : " + Keliling());  
    }  
}
```

```
class Lingkaran {  
    . . .  
    public String display() {  
        return ("Lingkaran"  
            + "\no jari-jari: " + getRadius()  
            + "\no Luas : " + Luas()  
            + "\no Keliling : " + Keliling());  
    }  
}
```

```
class TesPoligon {  
    public static void main(String args[])  
    {  
        // Deklarasi array  
        Poligon[] p = new Poligon[2];  
  
        // Add array poligon  
        p[0] = new SegiEmpat(17, 8);  
        p[1] = new Lingkaran(10);  
  
        // Display informasi  
        for (int i=0; i<p.length; i++) {  
            System.out.println(p[i].display());  
        }  
    }  
}
```

Method **display ()** yang dieksekusi ditentukan berdasarkan rujukan ke masing-masing objeknya

Contoh Polimorfisme

- Array yang berisi bangun geometri berbeda-beda

Kelas induk:

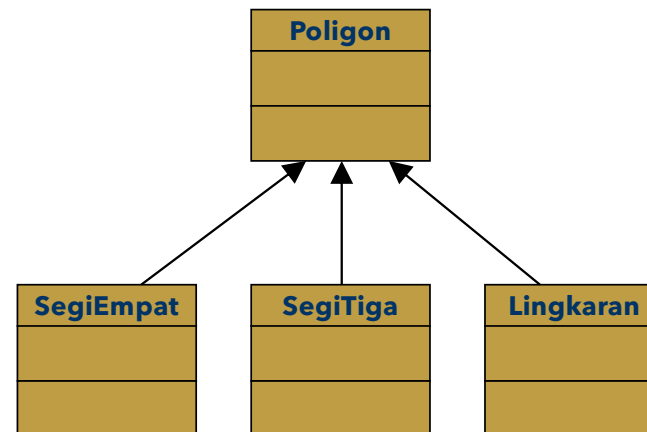
- [Poligon.Java](#)

Kelas anak:

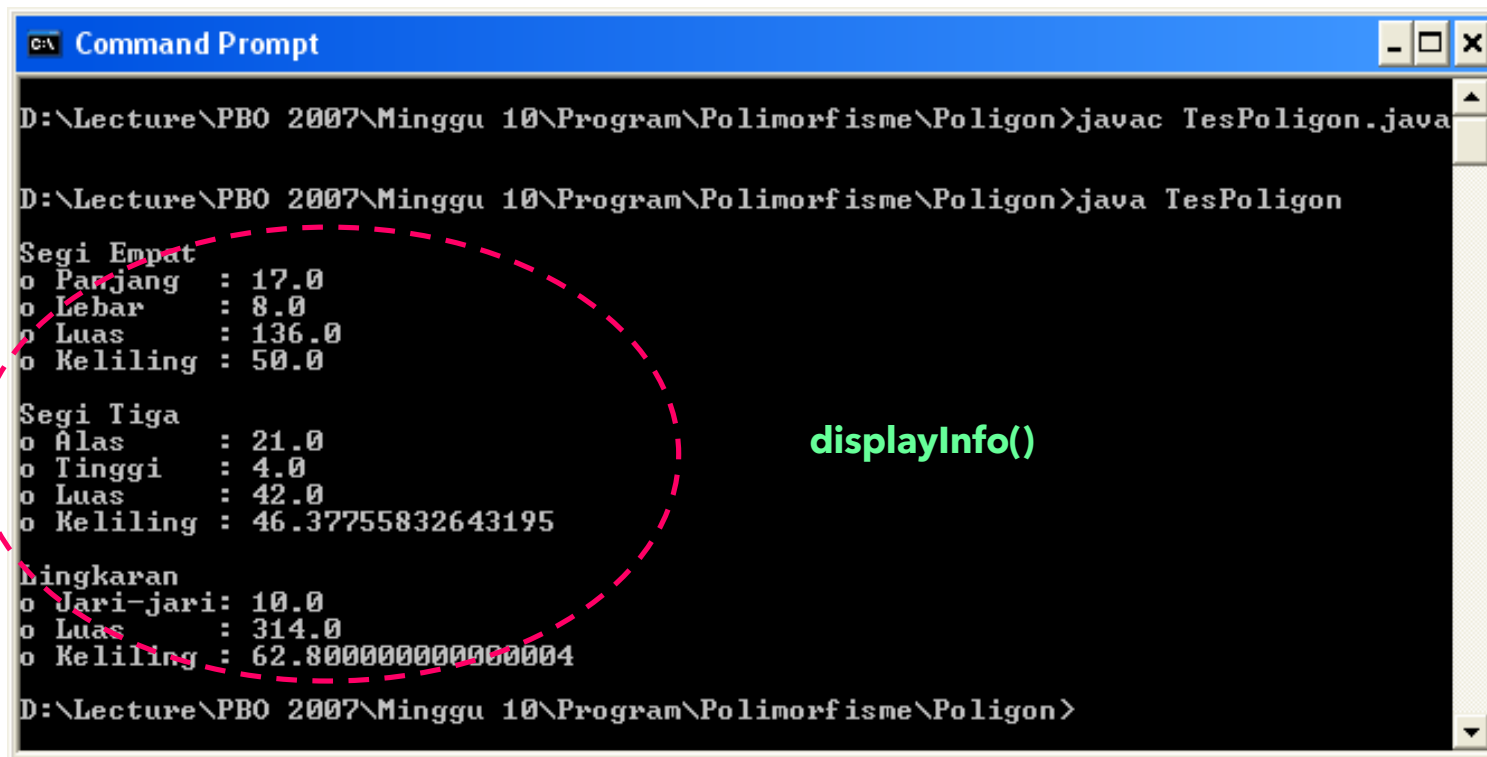
- [SegiEmpat.java](#)
- [SegiTiga.java](#)
- [Lingkaran.java](#)

Program utama:

- [TesPoligon.java](#)



Contoh Polimorfisme (lanjutan)



```
C:\ Command Prompt

D:\Lecture\PBO 2007\Minggu 10\Program\Polimorfisme\Poligon>javac TesPoligon.java

D:\Lecture\PBO 2007\Minggu 10\Program\Polimorfisme\Poligon>java TesPoligon

Segi Empat
o Panjang : 17.0
o Lebar : 8.0
o Luas : 136.0
o Keliling : 50.0

Segi Tiga
o Alas : 21.0
o Tinggi : 4.0
o Luas : 42.0
o Keliling : 46.37755832643195

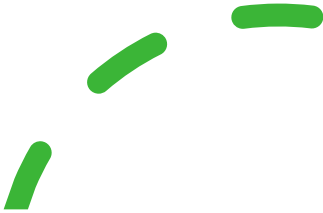
Lingkaran
o Jari-jari: 10.0
o Luas : 314.0
o Keliling : 62.800000000000004

D:\Lecture\PBO 2007\Minggu 10\Program\Polimorfisme\Poligon>
```

displayInfo()

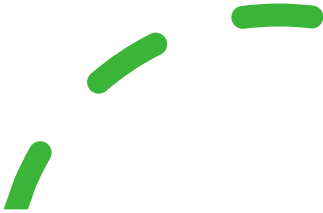


Tugas PBO

- Bentuk Kelompok Maksimal terdiri atas 2 orang.
 - Buat sebuah program OOP sederhana untuk suatu permasalahan (permasalahan ditentukan oleh kelompok) → Bahasa Java
 - Setiap kelompok membuat abstraksi keterkaitan dan deskripsi kelas yang dibutuhkan.
 - Permasalahan harus memuat kondisi pewarisan dan polymorphisme
- 



Tugas PBO

- Buat slide presentasi untuk tugas tersebut
 - Content:
 - 1) Permasalahan
 - 2) Abstraksi kebutuhan dan keterhubungan kelas (terdapat inheritance)
 - 3) Deskripsi atribut dan method yang dibutuhkan (beserta alasan penggunaan identifier → enkapsulasi)
 - 4) Contoh Penggunaan Polymorphisme
 - 5) Source code
 - 6) Screen shoot output program
- 



TERIMA KASIH

